Desarrollo de Aplicaciones de "In-door Marketing" utilizando Beacons BLE: Un reporte de Experiencia

Jenniffer Anzola Páez, Iván Rene Rodríguez Bedoya.
Universidad Piloto de Colombia, Bogotá, Colombia
Email japtheb@gmail.com, ivanrero@hotmail.com

***Abstract*—**
With the introduction of the Beacons by Apple in 2013, a new world of interaction started, especially in the area of marketing, where potential customers may receive information and obtain a better experience during their shopping. In the "In-door marketing", the location and preferences of the customers are used to present them products and offers. These systems may use Beacons BLE to locate the customers and look for the corresponding information. This paper presents our experience developing Bee Me, a software to manage Beacons in a store and provide information of products and offers to customers of a store. The paper focus on four main problems we must tackle: managing the information, locating the customers, presenting the information of products and offers in a map, and notifying users in their mobile devices. This paper also presents some tests we performed to assure the quality of our solution

Key words: --- Beacons, Indoor Marketing, and Indoor Location.

Resumen—
Con la introducción de los *Beacons* por Apple en 2013, un nuevo mundo de interacciones se ha iniciado, especialmente en el área de marketing donde clientes potenciales pueden recibir información y obtener una mejor experiencia durante sus compras. En el *Indoor Marketing* la ubicación y las preferencias de los clientes son usadas para presentar productos y ofertas. Estos sistemas pueden usar *Beacons* BLE para ubicar a los clientes y buscar la información correspondiente. Este artículo presenta nuestra experiencia desarrollando Bee Me, un software para administrar *beacons* y proveer información de productos y ofertas a los clientes de un almacén. Este artículo se enfoca en cuatro problemas que debimos enfrentar: el manejo de la información, la localización de los clientes, la presentación de la información de productos y ofertas en un mapa y la notificación a los clientes en sus dispositivos móviles. El artículo también presenta algunas pruebas realizadas para asegurar la calidad de la solución

Palabras Clave: --- *Beacons*, *Indoor Marketing*, Localización en interiores

I. Introducción

En el comercio tradicional, es muy importante para los almacenes y supermercados lograr que los clientes puedan encontrar fácilmente los productos y las promociones que son de su interés ya que en muchas ocasiones estos se pueden confundir en el momento de encontrar el producto o no enterarse a tiempo sobre las promociones de los artículos existentes.

En la actualidad, algunos almacenes están usando tecnologías de localización para orientar a sus clientes usando diferentes alternativas que permiten al usuario usar sus dispositivos personales para obtener información de su entorno. Los *Beacons* caen en esta categoría y permiten ubicar en las instalaciones de los almacenes puntos de contacto con los sistemas de la empresa siendo los teléfonos, *tablets* y computadores un mecanismo para conocer el entorno y establecer comunicación. [2]

Una aplicación puede usar estos *Beacons* para determinar la posición del cliente y orientarlo hacia los productos y las promociones que son de su interés.

El presente documento presenta nuestra experiencia en la implementación de Bee Me, un sistema uso *Beacons* como parte de una solución de *Indoor marketing* (Mercadeo en interiores).

En particular, durante el proceso de desarrollo se identificaron cuatro aspectos que requirieron ser resueltos para implementar este tipo de sistemas:

1- Integrar la información de las tiendas y las ubicaciones de los *Beacons* dentro de las mismas.
2- Obtener la información de proximidad de un *Beacon* a un móvil.
3- Ubicar al dispositivo móvil y los *Beacons* en un mapa que pueda presentarse al usuario.
4- Enviar notificaciones cuando el usuario se acerque a una sección o producto de su interés

II. CONTEXTO

Los *Beacons* (también conocidos como balizas) son dispositivos capaces de emitir una señal Bluetooth [1] de bajo consumo. Esta señal puede ser detectada por otros dispositivos, p.ej. un Smartphone, para determinar su proximidad.

A. Protocolos Desarrollados para los Beacons.

Los *Beacons* [3] son dispositivos que pueden transmitir datos a través de cualquiera de los dos perfiles de Bluetooth: El perfil GAP (Perfil de acceso genérico), que establece conexiones seguras, o el perfil GATT (perfil de atributos genéricos) que permite implementar protocolos manteniendo interoperabilidad con varios fabricantes. Los proveedores pueden implementar diferentes protocolos usando estos perfiles. Por ejemplo, el protocolo GAP *Advertising*, usando para hacer notificaciones y advertencias.

Algunos de los protocolos existentes en el mercado son: Apple *iBeacon* y Google *EddyStone.*

iBeacon

Apple[1] introdujo la tecnología BLE en el 2013, con la creación del protocolo "iBeacon". Este protocolo utiliza para la transmisión de datos un identificador único (UUID) el cual es captado por aplicaciones móviles capaces de interpretarlo. Adicionalmente, usando información extra, estas aplicaciones pueden ubicar el *Beacon* físicamente. En este esquema, cada *Beacon* puede variar en su potencia de transmisión. Apple provee una serie de librerías y kits de desarrollo (SDK) para poder usar estos dispositivos.

EddyStone

EddysStone es el protocolo para *Beacons* creado por Google[2]. A diferencia de *iBeacons*, este protocolo es de código libre y tiene soporte para múltiples plataformas, por ejemplo, plataformas como iOS y Android.

Google proporciona dos interfaces de desarrollo (dos APi): Nearby [7] y Proximity [8], los cuales brindan ayuda para la transmisión de datos hacia los equipos que se encuentren en el rango de los *Beacons*, así como poder monitorear el estado de los mismos.

B. Aplicaciones y Usos de los Beacons

Desde su introducción, los *beacons* han sido utilizados para una gran variedad de aplicaciones. Algunas de estas aplicaciones son, por ejemplo, *Indoor Marketing* (mercadeo en interiores), ayuda en Hospitales, y guías de turismo en Museos y sitios turísticos.

III. PROBLEMA

Desarrollar un sistema de mercadeo en interiores basado en *Beacons* presupone integrar la información de la ubicación de esos dispositivos la de los productos y las promociones en el almacén.

A. Caso de Estudio: Aplicación Bee Me

En este artículo se da cuenta de la experiencia de desarrollo de la solución Bee Me, una aplicación para ofrecer servicios de mercadeo en interiores a múltiples almacenes. Esto significa que es necesario administrar la información de los *Beacons*, productos y ofertas de varios almacenes al mismo tiempo.

El sistema se compone de las plataformas web Bee Me Admin, Bee Me Companies, Bee Me Products, el SDK BeeLibrary y la aplicación móvil Bee Market.

1. "Bee Me Admin"

Plataforma Web, donde el administrador podrá agregar los comercios, Empresas y/o Tiendas que contraten los servicios de "Bee Me", también se encargara de asignar los recursos a las empresas tales como Beacons y Administrar los usuarios que accederán a los servicios que ofrece "Bee Me".

Para efectos de los ejemplos, suponga que se tiene un almacén llamado *Merqueya*. Este almacén cuenta con sus propios *Beacons* y usuarios.

2. Bee Companies

Plataforma web, donde las empresas, comercios y/o tiendas inscritas a Bee Me podrán tener acceso y administrar los *Beacons* asignados por Bee Me Admin, crear promociones, cargar el mapa del establecimiento, asignar las posiciones de los *Beacons* dentro del mapa.

El acceso a Bee Companies se puede hacer a través de la URL *https://beecompanies.appspot.com/,* se ingresan las credenciales creadas y que son enviadas al correo del usuario creado para esta compañía.

3. Bee Me Products

Esta es la página del comercio inscrito a Bee Me, en donde está tenga registrados sus secciones y productos.

Para nuestro ejemplo, esto implica una página para *Merqueya***,** provista de algunas secciones y productos. Es necesario la creación de esta página ya que estas secciones y productos se consultarán a través de un SDK creado por "Bee Me" y se mostrarán en la aplicación móvil.

4. BeeLibrary

Librería creada por Bee Me en java, la cual se conecta mediante peticiones http a los *Endpoints*.

BeeLibrary trae la información asociada a la compañía del *backend*, las posiciones de los Beacons asignadas por *Merqueya***,** en Bee Companies, el mapa, los productos y secciones de está.

5. Bee Market

Aplicación móvil para Sistema Operativo Android, encargada de mostrar los productos, secciones, mapa y ubicación de los mismos al cliente del comercio en este caso *Merqueya***.**

B. Flujo de la información

[1] https://developer.apple.com/ibeacon/
[2] https://developers.google.com/beacons/

Se tiene un sistema que se divide en pequeños subsistemas (Bee Me, Bee Companies, *BeeLybrary*, App Móvil) por donde viajara la información es decir por las diferentes plataformas.

A continuación, se muestra un esquema general de cómo viaja la información a través de las plataformas.

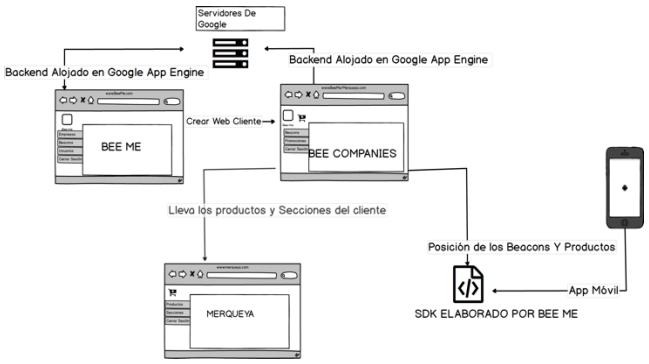

Fig. 1 Arquitectura Bee Me *Elaboración Propia*

En la Figura 1 que la plataforma web Administrativa, la plataforma web para la Compañía y la aplicación móvil se alojan en un mismo servicio: el Google App Engine [11]. En esta plataforma web se crea la empresa. Para el ejemplo la empresa creada se denomina *Merqueya*, le asignamos un usuario y los *Beacons* correspondientes.

La plataforma web da acceso a una empresa mediante el sistema de compañías, en la cual un usuario que haya sido previamente registrado podrá cargar el mapa de su Tienda, administrar los *Beacons* asignándoles productos y posiciones de referencia y crear y asignar *Beacons* a promociones.

Está información será accedida por los clientes de Merqueya a través de una aplicación Móvil en Android, la cual mediante una librería creada para plataformas Android ayudara con la generación de un mapa y sus elementos asociados y mostrará las promociones a un usuario cuando este cerca de un *Beacon*.

C. Principales Problemáticas

Durante el proceso de desarrollo se identificaron cuatro problemáticas principales a resolver:

1- Integrar la información de las tiendas y las ubicaciones de los *Beacons* dentro de las mismas.
2- Obtener la información de proximidad de un *Beacon* a un móvil.
3- Ubicar al dispositivo móvil y los *Beacons* en un mapa que pueda presentarse al usuario.
4- Crear notificaciones cuando el usuario se acerque a una sección o producto de su interés.

Las siguientes secciones detallan nuestras soluciones a cada una de estas problemáticas.

IV. Integrar la información de las tiendas y las ubicaciones de los Beacons

Para manejar la información de los diferentes almacenes, productos y promociones, se optó por una solución basada en los productos de Google para computación en la nube, herramientas tales como Google Cloud *DataStore*, *Entitys* y *Endpoints*.

A. Google Cloud DataStore

Google Cloud *DataStore* es una base de datos de documentos No SQL [10], que está construida para ser de alto rendimiento, de escalamiento rápido a conjuntos de datos grandes y de uso fácil para crear aplicaciones.

Para crear la aplicación, es posible establecer entidades que se guarden en la base de dato. Dado que la base de datos está orientada a documentos, estas entidades serán almacenadas como estructuras de datos sin validaciones o comportamiento definido en el sistema. En estas entidades, cada uno de los objetos o registros debe tener una llave única (un id) y un conjunto arbitrario de datos, donde cada dato puede ser de cualquier tipo (cadena de caracteres, numéricos, valores binarios). En nuestro caso, cada objeto se compone de un identificador y el mismo conjunto de datos definido en la entidad. La llave única, el id de cada entidad es asignado por el *Datastore.*

B. Entidades en Cloud DataStore

La interfaz de programación (el APi) *Datastore* permite guardar información en base a la definición provista en una clase Java. Esta clase debe tener la anotación *@Entity*. En tiempo de ejecución, la librería *Objectify* [15] usa estas anotaciones para identificar las entidades y manipular la base de datos. La figura 2 muestra un ejemplo con la entidad `Position` usando estas anotaciones.

```
import
com.googlecode.objectify.annotation.Entity;
import com.googlecode.objectify.annotation.Id;
import
com.googlecode.objectify.annotation.Index;

@Entity public class Position {

  @Id @Index private Long id;
  @Index private Long beaconId;
  @Index private Long companyId;
  private String sectionId;
  private String productId;
  private Integer x;
  private Integer y;

  public Long getId() {
    return id;
  }
  public void setId(Long id) {
    this.id = id;
  }
  public Long getBeaconId() {
    return beaconId;
  }
  public void setBeaconId(Long beaconId) {
    this.beaconId = beaconId;
  }
  public Long getCompanyId() {
    return companyId;
  }
  public void setCompanyId(Long companyId) {
    this.companyId = companyId;
  }
  public String getSectionId() {
    return sectionId;
  }
  public void setSectionId(String sectionId) {
    this.sectionId = sectionId;
  }
  public String getProductId() {
    return productId;
  }
  public void setProductId(String productId) {
    this.productId = productId;
  }
  public Integer getX() {
    return x;
  }
  public void setX(Integer x) {
    this.x = x;
  }
  public Integer getY() {
    return y;
  }
  public void setY(Integer y) {
    this.y = y;
  }
}
```

Fig. 2 Definición Entidad Position

En la Figura 2 se puede notar como, dentro de la clase, se declaran las propiedades o atributos de la entidad: El atributo *id,* adicional del tipo de dato tiene la anotación @Id para indicarle al *Datastore* que este será la llave única. Los atributos *beaconId* y *companyId* tienen la anotación *@Index* para que el atributo sea indexado. Las aplicaciones pueden usar estos índices para crear filtros y búsquedas sobre la entidad.

Además de las clases entidad, es necesario crear otras clases que realicen operaciones sobre esas entidades. La Figura 2 muestra una clase `Positions` que contiene operaciones para crear, modificar y borrar entidades de tipo `Position`

```
public class Positions {
  static {
    register(Position.class);
  }
  public static void create(Position position, long
companyId) {
    position.setCompanyId(companyId);
    ofy().save().entity(position).now();
  }
  public static void edit(Position position) throws N
otFoundException {
    Position              foundPosition              =
position(position.getId());
    foundPosition.setBeaconId(position.getBeaconId());

foundPosition.setCompanyId(position.getCompanyId());
    foundPosition.setX(position.getX());
    foundPosition.setY(position.getY());

foundPosition.setSectionId(position.getSectionId());

foundPosition.setProductId(position.getProductId());
    ofy().save().entity(foundPosition).now();
  }
  public static void delete(Long positionId)
throws NotFoundException {
    ofy().delete().entity(position(positionId)).now();
  }
}
```

Fig. 3 Clase Positions en Java Usando la librería Objectify

En la figura 3 se puede observar el método `create` de la clase `Positions` donde se definen los parámetros de entrada los cuales son el objeto position y el *companyId*, se setea el *companyId* a la position para posteriormente llamar la librería Objectify y el método sabe, con el cual guardamos la posición creada, en el objeto position y finalmente llamamos al método now () para indicarle a la librería que este se ejecute inmediatamente.

Con Objectify no solo podemos obtener las operaciones básicas (create, update, delete) también podemos hacer uso de los métodos load (para obtener un valor o un listado de valores), cast (), cache (), notify () entre otros.

C. Endpoints

Los *EndPoints* son puntos de envío y recepción de mensajes de acuerdo con la arquitectura REST.

Para el desarrollo del proyecto se ha optado por usar "*Cloud EndPoints*" [10], está es una tecnología de Google para exponer fácilmente las URL y los servicios REST.

El objetivo de crear *EndPoints* en el *backend* de la plataforma es exponer la información almacenada en la base de datos de una manera estándar, siendo APi REST una alternativa para definir la comunicación con los diferentes clientes.

La creación de un *EndPoint* se hace usando la anotación @APi dentro de esta se nombra el APi, la versión, *namespace* en este atributo se asigna la anotación *@APiNamespace*, para que el APi tenga el nombre especificado de lo contrario tomara el nombre por defecto que sería el nombre `com.appspot.nombre-del-proyecto-id.nombreAPi`.

Algunos atributos existentes en las declaraciones nos permiten definir el tipo de API que se quiere usar asi por ejemplo la anotación @APi (y sus respectivos valores: *ownerDomain*, *ownerName*, *packagePath*) dan una configuración general del api que se esta definiendo.

Usando anotaciones sobre métodos se pueden definir puntos de acceso distintivos, como es el caso de la clase *PromotionsEndPoints*, en esta vemos las diferentes anotaciones @APIMethod que permiten declarar cada uno de las acciones del API actual. La Figura 3 muestra como ejemplo el API de promociones.

```
@APiMethod(name = "create", httpMethod = POST)
public void create(HttpServletRequest request,
Promotion promotion) throws NotFoundException {
  Users.validateToken(request.getHeader("X-Token"));
  Promotions.create(promotion);
}
```

Fig. 4 Método create de la clase PromotionsEndpoints

En la Figura 4 puede verse que dentro del método se hace una validación de *token*, el cual se crea cuando el *Entity* de sesión se asigna al usuario y se usa el id del objeto que es generado aleatoriamente como *token*, si este es válido se hace el llamado al método create de la clase `Promotions`.

V. Obtener la información de proximidad de los Beacons a un móvil

La aplicación como tal no hace detección de la señal generada por el *Beacon*, esta detección se hace por medio del SDK proporcionado por la plataforma que en conjunto con otras librerías realizan todo el trabajo para hacer las peticiones web necesarias para obtener la información actual. En el listado de dependencias en el archivo **build.gradle** el proyecto en Android Studio se deberá solamente incluir la librería del sistema (Ver Figura 4).

```
dependencies {
  compile project(path: ':BeeLibrary')
}
```

Fig. 5 Dependencias del Archivo build.gradle

La librería de Estimote tiene algunas clases necesarias para que podamos interactuar con los *Beacons* como la clase `BeaconManager` quien es nuestra puerta de entrada para poder hacer uso de las características de los *Beacons* tales como `Ranging` y `Monitoring [15]` necesarias para detectar los *Beacons*.

Los *Beacons* nos proporcionan tres valores

- UUID: representado por una cadena de caracteres (ejemplo: "B9407F30-F5F8-466E-AFF9-25556B57FE6D").
- *Major number*: Por lo general es un número entero comprendido entre 1 y 65535, es un identificador para marca un área, ejemplo, todos los *Beacons* en un determinado piso o salón.
- *Minor number*: Al igual que el *major number* es un valor entero, este identificador distingue un *Beacon* dentro de un grupo de *Beacons*.

Con estos 3 valores podemos determinar la detección de uno o varios *Beacons* por ejemplo podemos indicarle a nuestra aplicación que detecte la señal de un *Beacon* por su UUID, o si deseamos ser más específicos podemos hacer las siguientes combinaciones, UUID + *major number* + minor *number, UUID* + *major number*, UUID únicamente o podemos no determinar parámetros y por tanto detecta todos los *Beacons* en el rango de alcance que estos tengan [9].

Una de las características que nos brindan los *Beacons* son las regiones, estas son básicamente los rangos o el área a la que puede estar el *beacon*.

- Inmediata: Es decir el usuario se encuentra a muy pocos centímetros del *Beacon*.
- Cercana: El usuario se encuentra a menos de 10 metros del *Beacon* detectado.
- Lejana: El *Beacon* está a más de 10 metros del usuario que percibe la señal.

Fig. 6 Distancia y Proximidad [16]

En la Figura 6 que al estar a un metro del *Beacon* que emite la señal, la potencia de la señal es captada al máximo de la potencia emitida, y que, al alejarse el teléfono, aunque capta la señal está va a ser menor, esto se define mediante la lectura del valor RSSI del *Beacon*, el cual es el indicador de la fuerza de la señal, el valor RSSI es mínimo de -30 db (decibeles).

Entre más cerca se encuentre el dispositivo al *Beacon* que emite la señal, esta será más intensa y lo muestra en la región correspondiente.

En la figura 7, se muestra la relación existente entre distancia del dispositivo contra la potencia de emisión de la señal.

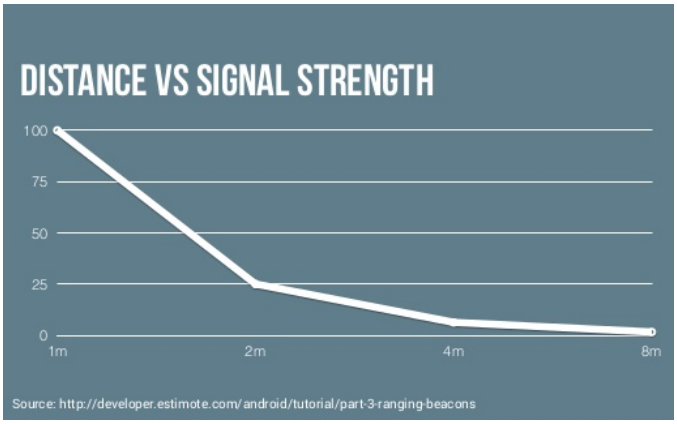

Fig. 7 Distancia Vs Potencia de la Señal del Beacon [11]

La relación decibeles valor RSSI se puede ver en la Tabla 1

Distancia del *Beacon*	**Valor RSSI**
1 metros	-40 decibeles
2 metros	-46 decibeles
4 metros	-52 decibeles
8 metros	-58 decibeles
16 metros	-64 decibeles

Tabla 1 Relación Distancia Beacon" / valor RSSI

VI. Ubicar al dispositivo móvil y los *Beacons* en un mapa que pueda presentarse al usuario

La plataforma web permite al cliente hacer un cargue del mapa de la tienda y con este ir ubicando los *Beacons* asociados mediante las relaciones existentes en la base de datos. Existen conexiones entre las entidades de los Productos, Promociones y los *Beacons*, permitiendo exponer la información mediante un API REST, la cual será consumida por la aplicación móvil.

A. Cargue del Mapa en la Web de Bee Companies

Se utilizó una directiva de angular llamada *ng-file-upload*, el proceso de cargue de la directiva a nuestro proyecto se puede consultar en la documentación disponible en *GitHub*, añadir el módulo en las dependencias de nuestro archivo y en nuestro *Dataservice* hacer los llamados mediante Ajax a los *Endpoints* correspondientes.

El proceso de carga de un archivo se separa en dos partes: la generación de una url segura donde enviar el archivo y enviar el fichero mediante una petición POST con la dirección relacionada en el primer paso. Estos dos pasos se realizan desde la aplicación web mediante peticiones a los a la compañía.

Adicionalmente se comprueba que el token de la sesión sea válido y mediante un tag de imagen en la aplicación web se mostrará el archivo subido.

B. Ver el mapa en la Aplicación móvil

Para poder presentarle al usuario el mapa que la empresa ha cargado en la plataforma de compañías y que este lo pueda visualizar en la app móvil ha sido necesario hacer uso de la librería Picasso [5] [13], la cual se encarga de la descarga de la imagen, el manejo de la cache y manejo de memoria.

Además, de ayudar a corregir errores que pueden ocurrir durante la descarga tales como

- Administrar el reciclado de los *ImageView* y cancelar la descarga en un *adapter*.
- Transformación compleja de imágenes con un uso de memoria mínimo.
- Caching automático de memoria y disco.

C. Calcular la posición del Usuario, calculando el Centro de gravedad de un Triángulo

Para poder hacer este cálculo como mínimo debemos tener 3 *Beacons*, los cuales trazarán un área y dentro de esta área asumimos se encuentra el usuario, con los 3 *Beacons* más cercanos se traza el área triangular, se calcula el centro de masa y usando la fuerza de la señal del *Beacon* se estima una posición más acertada. Ver figura 7.

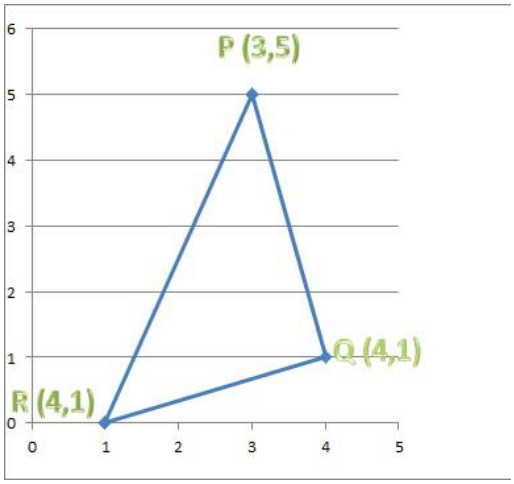

Fig. 8 Calcular Centro de Gravedad de un Triángulo [20]

En la figura 8 tenemos un ejemplo donde se determinan las coordenadas de los tres vértices del triángulo PQR, el punto P tiene las coordenadas (3, 5), el punto Q (1,4) y R (1, 0).

Luego se debe sumar los valores de las coordenadas en x, se deben agregar las tres coordenadas de los puntos P, Q, R. Después sumar las 3 coordenadas de y.

Por último, se calcula el promedio de las coordenadas en (x, y). Estas coordenadas corresponden al centro de gravedad del triángulo, también conocido centro de masa.

Para calcular el promedio, se divide la suma de las coordenadas por 3 y se calcula el centro de gravedad en el triángulo.

No importa qué lado se elija el centro de gravedad siempre estará en el mismo punto. (Ver figura 9)

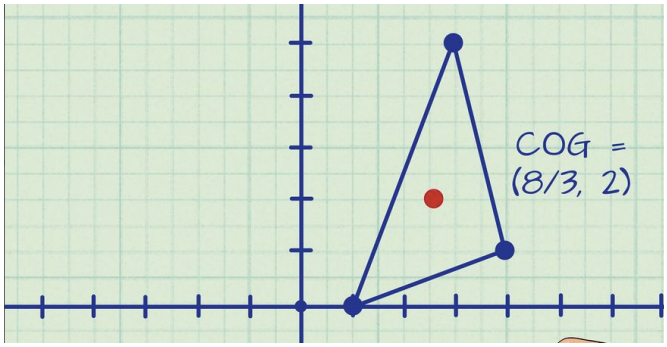

fig. 9 Centro de Gravedad de un Triángulo [20]

De esta manera se traza el centro de gravedad en el triángulo. El centro de gravedad, o centroide, es el promedio de las coordenadas x y como se observa en la figura 8.

VII. Enviar notificaciones cuando el usuario se acerque a una sección o producto de su interés.

Una empresa que use la plataforma web puede crear promociones y asignar el *Beacon* que lanzará la notificación, una vez el usuario esté cerca el SDK que se usa en la aplicación móvil seguirá una serie de comprobaciones con los *Beacons* y con el *backend* a través del API REST para chequear la existencia de una promoción y poder mostrarla al usuario.

La notificación que está asignada al Beacon que el dispositivo encontró, el usuario puede dar clic sobre la notificación lo cual abrirá la aplicación en el detalle de la promoción.

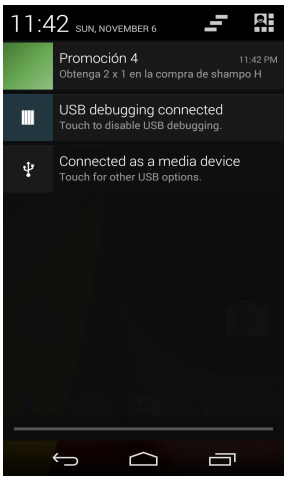

fig. 10 Elaboración Propia.

En la figura 10 se muestra la notificación que recibe el móvil y se visualiza en el panel de notificaciones de Android.

Una vez se abre la aplicación para mostrar la notificación que ha llegado al celular, la alerta de notificación desaparece.

A. Manejo de Notificaciones en BeeLibrary

El proceso de verificación de una promoción se caracteriza por el uso del servicio Android que funciona mientras la aplicación está en segundo plano y va revisando continuamente los *Beacons* que detecta, al ser más de una opción las disponibles solo se usa el *Beacon* más cercano para hacer el chequeo y no mostrar múltiples notificaciones en un corto periodo de tiempo.

Al llegar a un *Beacon* el SDK lo detecta y con el id resultante se hace una consulta al API web que entrega la información de la promoción asociada y si esta existe se genera la notificación de la promoción que el usuario de la app móvil puede ver en su celular.

VIII. RESULTADOS

Para probar la correcta implementación del Software se crearon 4 escenarios:

A. Precisión en distancias diferentes

Para esta prueba se escogió un recinto de la Universidad Piloto de Colombia, en el edificio APR, el salón 305 que tiene una medida de 7 x 7 metros cuadrados, se tomaron distintas distancias para estimar el porcentaje de error que se tiene al estar más lejos los *Beacons* del dispositivo móvil (Ver Figura 9).

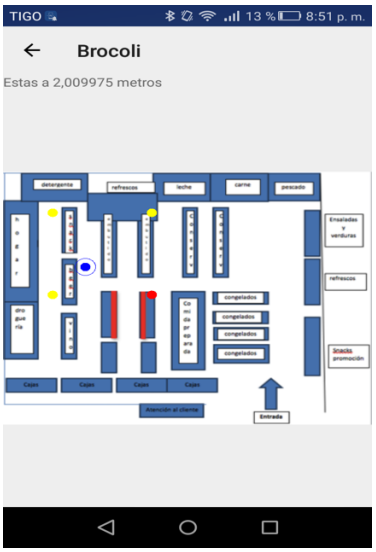

Fig. 11 Resultado a una distancia de 3 metros del Beacon

En la figura 11 el dispositivo móvil nos indica que estamos a una distancia de 2.1 metros aproximadamente cuando la medición física fue de 3 metros indicándonos que existía un margen de error de 0.9 metros.

Para está prueba se realizaron dos mediciones los resultados de estas se pueden apreciar en las figuras 10 y 11.

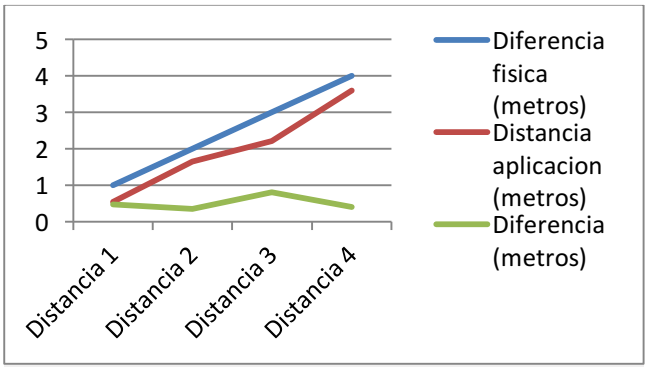

Fig. 12 Resultados Prueba de distancia del dispositivo al "Beacon" primera medición.

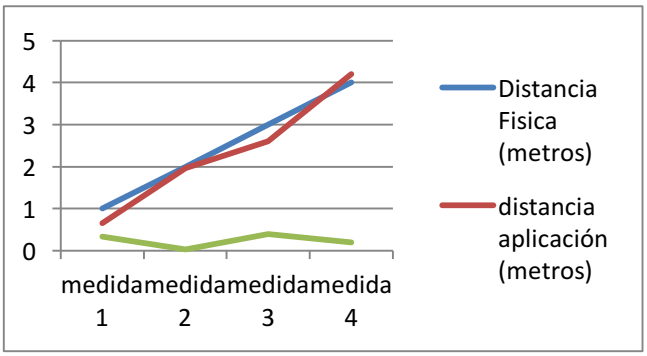

Fig. 13 Resultados Prueba de distancia del dispositivo al Beacon segunda medición.

En la figura 11 nos indican que los valores de la distancia que el dispositivo nos muestra con relación a las distancias físicas pueden variar aun cuando la distancia física siempre sea la misma, ya que el algoritmo utilizado siempre ubica al usuario en medio de tres puntos en este caso de los *Beacons* y hace un promedio de estas distancias para ubicarlo.

En la figura 13 se toman las mismas distancias físicas que en la figura 11 y el resultado obtenido fueron distancias diferentes a las arrojadas en la primera toma de datos.

B. Precisión del Cálculo de la Señal con Obstáculos

Si en el entorno en el que se va a desplegar la aplicación existen obstáculos como paredes y puertas se debería poder saber cuál es la proporción de error, las pruebas con obstáculos serán de ayuda para resolver errores si se realizan próximas versiones del algoritmo. Para la prueba en un escenario con obstáculos se tomó medidas en el salón 304 del edificio APR de la Universidad Piloto de Colombia y el pasillo del mismo piso con una pared de por medio se tomaron los valores a varias distancias los resultados se pueden apreciar en la tabla 2.

Distancia física	Distancia dispositivo
3 metros	1.97 metros
4 metros	0.7 metros
5 metros	NaN (Valor indefinido)

Tabla 2 Medición con obstáculos entre el Beacon y el dispositivo Móvil.

En la tabla 2 se muestran los valores recogidos con la prueba y se evidencia como la pared hace que la señal no sea interpretada de manera correcta por el celular, hasta el punto de en una de las mediciones no lograr calcularla.

Con la realización de esta prueba se pudo notar que si se interponen paredes y una distancia lo suficientemente grande el dispositivo no podrá hacer el cálculo de la distancia entre él y el *beacon*, ya que la pared hace que la señal que el *beacon* genera se atenué y esto genera distorsión en la señal. Cuando está a distancia corta el margen de error es más grande debido a que la señal pierde potencia.

C. Precisión en Movimiento

Uno de los escenarios más importantes para el sistema es cuando el dispositivo se encuentra en movimiento, ya que para llegar a un producto o una sección de productos los usuarios van a estar en movimiento, por esta razón se realiza esta prueba, para encontrar el porcentaje de error de la señal entre el dispositivo móvil y el *Beacon* al que se acerca. Esta prueba se realizará dentro del salón 304 en el edificio APR de la Universidad Piloto de Colombia. (Ver tabla 3)

Velocidad	Distancia física	Distancia dispositivo
0.1 Kph	2 metros	1.66 metros
0.1 Kph	3 metros	2.2 metros
0.1 Kph	4 metros	2,55 metros
0.1 Kph	5 metros	3.2 metros

Tabla 3 Mediciones tomadas en movimiento.

En la Tabla 3 la distancia del dispositivo varia con el movimiento siendo en algunas ocasionas muy imprecisa y generando un margen de error mucho mayor.

El porcentaje de error para esta prueba fue del 81%.

D. Precisión de las notificaciones

En este escenario se probará él envió de notificaciones cuando se encuentran varias promociones cerca. Estas pruebas se realizaron para verificar el comportamiento de las notificaciones cuando hay varios bancos cercanos y que el motivo de que existan varios cerca no genere conflictos.

Esta prueba se realizó en el edificio APR salón 304, Los resultados se pueden apreciar en la Tabla 5.

Distancia física	Distancia al dispositivo Beacon 1	Distancia al dispositivo Beacon 2	Distancia al dispositivo Beacon 3	Beacon que envía la notificación Primero
1 metro del Beacon 1	1.2 metros	2.5 metros	3 metros	Beacon 1
1 metro del Beacon 2	2.3 metros	1.5 metros	3 metros	Beacon 2
1 metro del Beacon 3	3.6 metros	2.7 metros	1.02 metros	Beacon 3
2 metros del Beacon	1.9 metros	1.8 metros	1.85 metros	Beacon 2 Beacon 3 Beacon 1

Tabla 4 Envió de Notificaciones al Celular.

En la Tabla 4 se muestran los resultados obtenidos en la realización de las pruebas realizadas para validar él envió de las notificaciones nos arrojaron resultados diferentes en cuanto el orden de envió de estas, ya dicho orden depende de la potencia de la señal que el dispositivo detecte, entre más fuerte la señal del *Beacon* el dispositivo lo detecta y revisa si este tiene asignada alguna notificación y muestra, luego sigue con la siguiente más fuerte y así sucesivamente.

IX. Conclusiones

Los *Beacons* no solo sirven para localizar objetos, también para enviar y recibir datos hacia otros dispositivos.

En cuanto al sistema creado se concluye puede mejorarse la forma de asignar las posiciones desde Bee Companíes en el mapa que el almacén cargue.

En cuanto a la distancia de los *Beacons* se concluye que cuando el dispositivo Móvil no se encuentra en movimiento los resultados fueron favorables arrojando un margen de error de aproximadamente 20 a 30 cm.

El uso de la aplicación en movimiento, aunque recalcula la posición del usuario y lo reubica en el mapa genera un error más grande debido a que siempre va a ubicar al dispositivo móvil en el medio de los tres puntos más cercanos.

Aunque los *Beacons* no necesitan el consumo de datos para enviar información, la aplicación móvil si se conecta a los servicios REST que están en el servidor, por tanto, necesita usar Datos para descargar los recursos tales como el mapa y las posiciones asignadas a los *Beacons*, para presentarle dicha información al cliente del almacén.

En un entorno con obstáculos, como paredes o puertas, los resultados no tendrán un alto porcentaje de precisión, ya que causan variaciones en la potencia de la señal generando una atenuación la misma

Los resultados de las pruebas realizadas sobre las notificaciones, fueron favorables ya que dependían de la potencia de la señal que el *Beacon* emite.

X. Referencias

.

[1] S.A Gonzales Vergara “Tecnología Bluetooth”. Instituto Politécnico Nacional.

[2] D. Mora Salcedo, D. A Apolinar Santos. “BEACON CITY”. Universidad Católica de Colombia.

[3] M. E Bentos Vera “Beacons”. Universidad de la Republica (Uruguay).

[4] L. Padrón. “BulletPoint”. Tesis de Grado. Universidad de la Laguna.

[5] N. Peitek, M. Póhis. “Picasso Easy Image Loading on Android”. Future Studio

[6] F. Xia. “A localization strategy based on *n*-times trilateral centroid with weight”. Dallas University of Technology.

[7] Nearby. [online]. Disponible en https://developers.google.com/nearby .

[8] Proximity Beacon APi. [en línea]. Disponible en https://developers.google.com/beacons/proximity/sharing.

[9] Google Cloud Datastore Overview. [online]. Disponible en https://cloud.google.com/datastore/docs/concepts/overview

[10] Cloud Endpoints. [online]. Disponible en https://cloud.google.com/endpoints

[11] Google App Engine Documentation. [online]. Disponible en https://cloud.google.com/appengine/docs

[12] Photo View. [En línea]. Disponible en *https://github.com/chrisbanes/PhotoView*

[13] Picasso. [online]. Disponible en http://square.github.io/picasso

[14] What are region Monitoring and Ranging? [online]. Disponible en https://community.estimote.com/hc/en-us/articles/203356607-What-are-region-Monitoring-and-Ranging .

[15] Objecty Lybrary. [Online]. Disponible en https://github.com/objectify/objectify/wiki

.